

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ «КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені Ігоря Сікорського»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

**Кафедра системного програмування та спеціалізованих комп’ютерних систем**

**Лабораторна робота №2**

з дисципліни **Бази даних і засоби управління**

*на тему: “Проектування бази даних та ознайомлення з базовими операціями СУБД PostgreSQL”*

Виконав:

студент ІII курсу

групи КВ-91

Тегза Д. І.

Перевірив:

Павловський В. І.

Київ – 2021

# Постановка задачі

*Метою роботи* є здобуття вмінь програмування прикладних додатків баз даних PostgreSQL.

*Загальне завдання* роботи полягає у наступному:

1. Реалізувати функції перегляду, внесення, редагування та вилучення даних у таблицях бази даних, створених у лабораторній роботі №1, засобами консольного інтерфейсу.
2. Передбачити автоматичне пакетне генерування «рандомізованих» даних у базі.
3. Забезпечити реалізацію пошуку за декількома атрибутами з двох та більше сутностей одночасно: для числових атрибутів – у рамках діапазону, для рядкових, – як шаблон функції LIKE оператора SELECT SQL, для логічного типу – значення True/False, для дат – у рамках діапазону дат.
4. Програмний код виконати згідно шаблону MVC (модель-подання- контролер).

**Інформація про програму**

Використана мова програмування: Python 3.10.

Використані бібліотеки: psycopg2 (для зв’язку з СУБД), time (для виміру часу запиту пошуку, що у 3 завданні)

Використаний шаблон проектування: MVC.

**Опис предметної галузі**

Для даної лабораторної роботи я вибрав тему – Гуртожитки КПІ.

При проектуванні бази даних «Гуртожиток» можна виділити наступні сутності: загальні відомості про гуртожиток (dormitory), відомості про кімнату (room), відомості про тип кімнати (room\_type), студент або жилець (inmate).

Атрибути заданих сутностей:

* + - 1. dormitory: number, type, address, faculty.
      2. room: number, floor, settlement, eviction
      3. room\_type: seats, furniture, gender.
      4. inmate: name, surname, gender, study\_year.

**Опис зв’язків**

У конкретному гуртожитку находиться багато кімнат, тому між сутностями dormitory і room зв’язок 1:N

Певному типу кімнат можуть відповідати зразу багато кімнат, тому між сутностями room\_type і room зв’язок 1:N

В одній кімнаті можуть проживати декілька студентів, тому між сутностями room і inmate зв’язок 1:N

Певна прибиральниця відповідальна за багато кімнат, також за одну кімнату відповідальні декілька прибиральниць (в різні дні).

**Відомості про предметну галузь з лабораторної роботи №1**

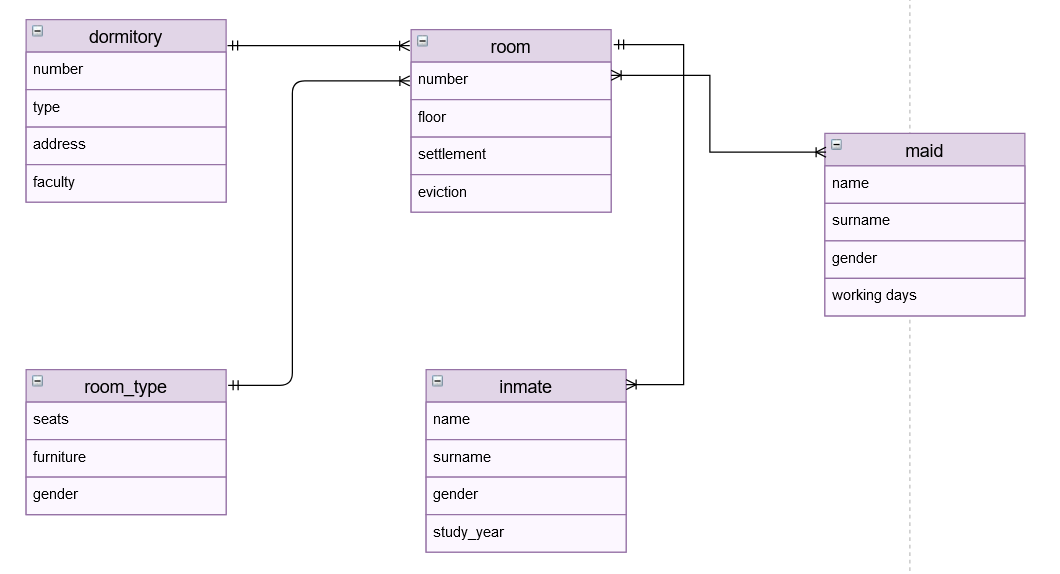


Рисунок 1 - ER-діаграма побудована за нотацією “Пташиної лапки (Crow’s foot)”, задана ER-діаграма була побудована у додатку [draw.io](https://app.diagrams.net/)

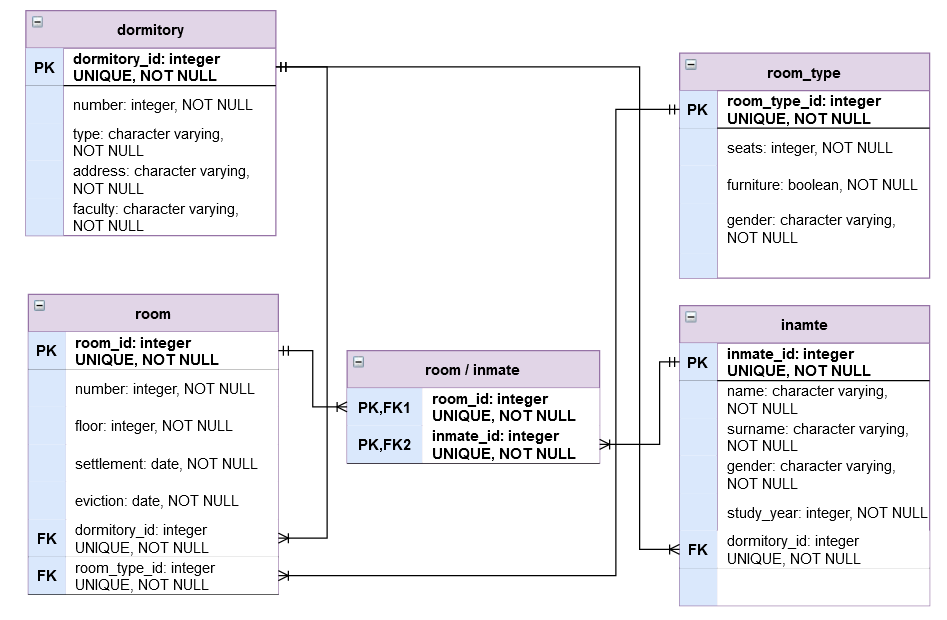


Рисунок 2 - Схема бази даних, побудовано у додатку [draw.io](https://app.diagrams.net/)

**Таблиця 1 - Опис структури БД.**

|  |  |  |
| --- | --- | --- |
| Відношення | Атрибут | Тип атрибуту |
| **dormitory** –містить дані про готель | **dormitory\_id** – унікальний ідентифікатор **number** – номер гуртожитку **type** – тип (блочний/не блочний) **address** – адреса **faculty** – факультет, який поселяють у даний гуртожиток | Integer integer character varying character varying character varying |
| **room\_type** – містить дані про тип кімнати | **room\_type\_id** – унікальний ідентифікатор **seats** – кількість місць у кімнаті **furniture** – кімната здається разом із меблями, чи без **gender** – жіноча кімната або чоловіча | integer integer boolean character varying |
| **room** – містить дані про кімнату | **room\_id** – унікальний ідентифікатор **number** – номер кімнати **floor** – поверх **settlement** – дата поселення **eviction** – дата виселення **dormitory\_id** – ідентифікатор гуртожитку **room\_type\_id** – ідентифікатор типу кінмати | integer integer integer date date integer integer |
| **inmate** – містить дані про студента (жильця) | **inmate\_id** – унікальний ідентифікатор **name** – ім’я студента **surname** – прізвище **gender** – стать **study\_year** – курс (рік навчання) **dormitory\_id** – ідентифікатор гуртожитку | integer character varying character varying character varying integer integer |
| **room/inmate** – відношення жильців до кімнат | **room\_id** – ідентифікатор кімнати **inmate\_id** – ідентифікатор жильця | integer integer |

**Структура програми**

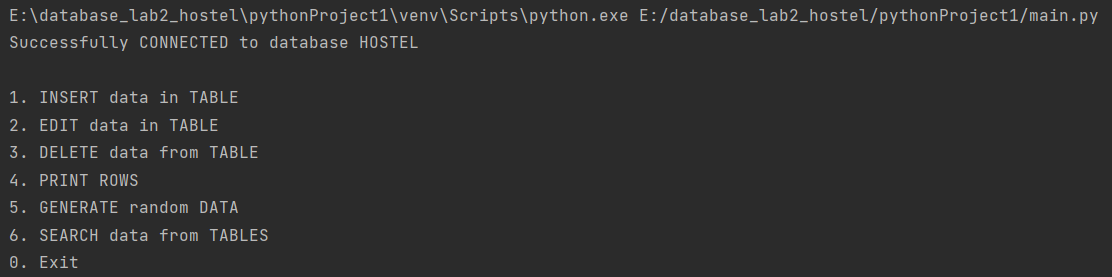
За шаблоном проектування MVC, додаток складається з таких модулей:

*model.py* — підключається до БД та виконує операції – SELECT, INSERT, DELETE, UPDATE та, більш складні операції з БД.

*controller.py* — головне та допоміжні меню, для зручного керування БД.

*mainl.py* — точка входу в програму, та підключення до БД.

# Схема меню користувача



На знімку екрану термінала продемонстровано початкове меню, де можна побачити функції які можна виконати з БД. Кожна команда запускає відповідну функцію з файлу controller.py, яка в подальшому передає аргументи у функцію файлу model.py, яка в свою чергу формує і здійснює запит до бази даних.

***Методи реалізовані до пункту 1 завдання лабораторної роботи:***

1. INSERT data in table — викликає функцію вставки даних у таблицю бази даних;
2. EDIT data in table — викликає функцію редагування даних у таблиці бази даних;
3. DELETE data from table — викликає функцію видалення даних у таблиці бази даних;
4. PRINT rows — викликає функцію виводу даних з таблиці бази даних.

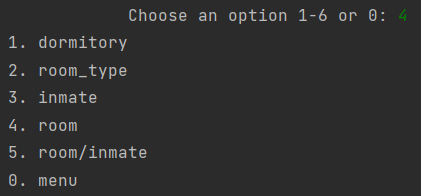
***Метод реалізований до пункту 2 завдання лабораторної роботи:***

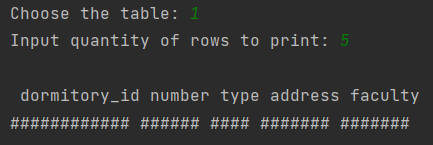
1. GENERATE random data — дозволяє користувачеві заповнити таблицю, або всю базу даних випадково згенерованими даними.

Кожна з описаних вище функцій викликає допоміжну функцію select\_table(), даний скрипт дозволяє користувачеві обирати таблицю

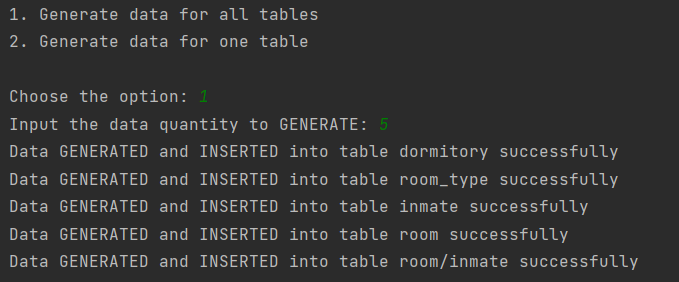
***Метод реалізований до пункту 3 завдання лабораторної роботи:***

1. SEARCH data from tables — викликає функцію пошуку даних у таблицях за атрибутами та поєднання таблиць за ключем.

Виведення таблиці на екран

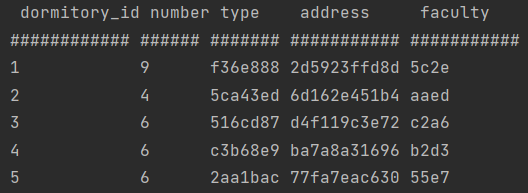


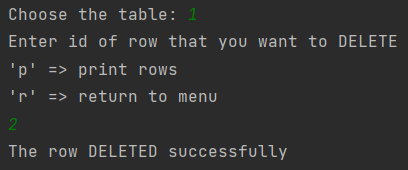
Оскільки ця таблиця ще не заповнена, давайте її заповнимо псевдо випадковими даними



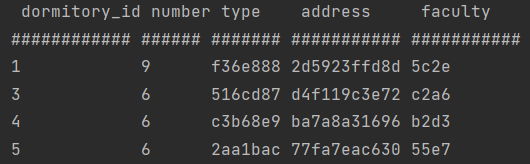
Запит на видалення

Таблиця **dormitory** до видалення даних:

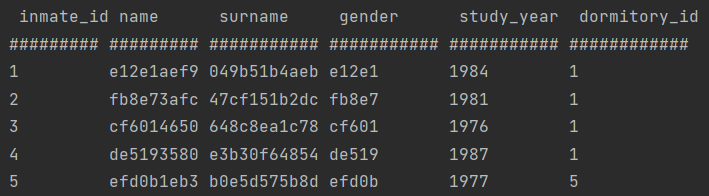


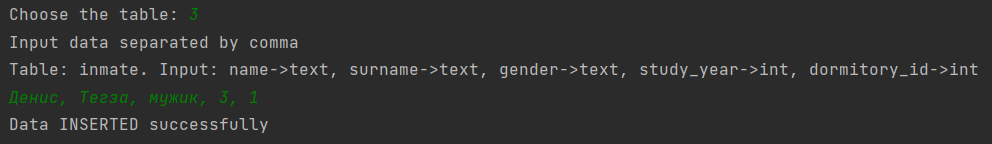


Таблиця **dormitory** після видалення даних:

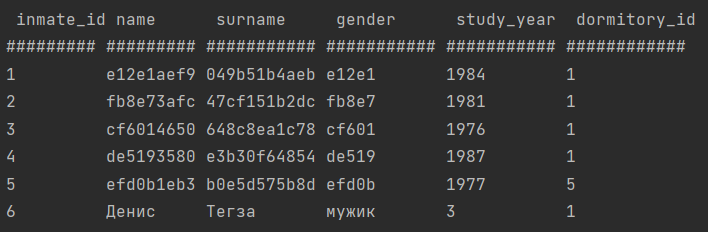
Запит на вставку поля

Таблиця **inmate** до вставки даних:

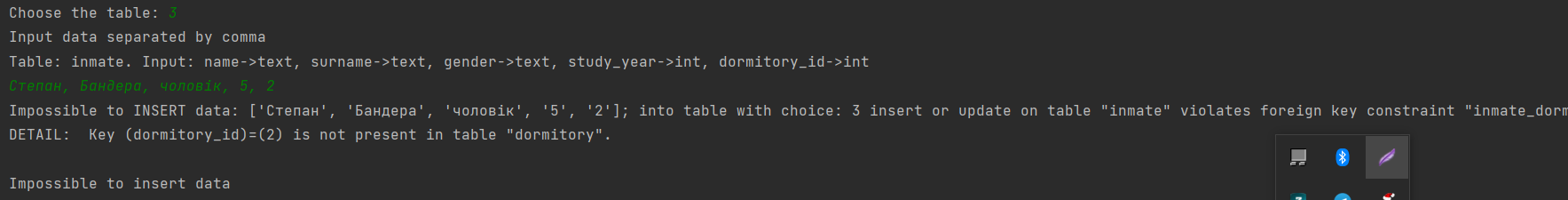




Таблиця **inmate** після вставки даних:



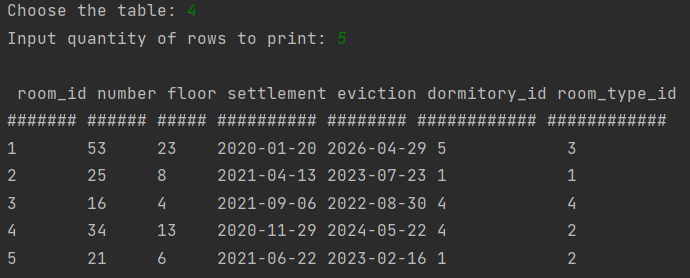
Давайте провіримо, як відреагує програма, якщо в поле **dormitory\_id** ми введемо неіснуючий **id**:

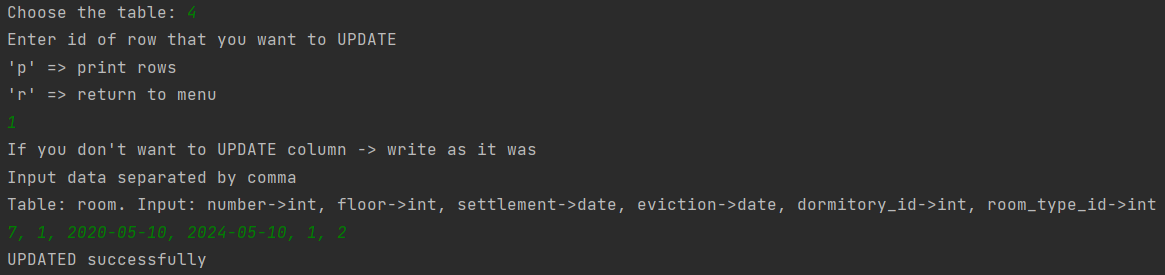


Як бачимо, при посиланні на неіснуючий **id** нам вибиває помилку

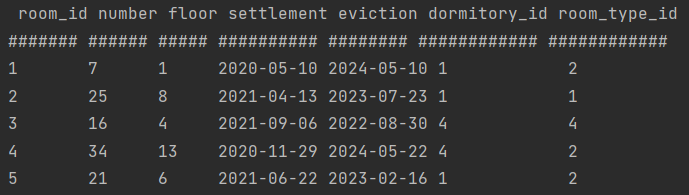
Запит на редагування поля

Таблиця **room** до редагування рядка





Таблиця **room** після редагування рядка



Код програми

main.py

**import controller  
import model  
  
model.connect()  
  
controller.menu()  
  
model.disconnect()**

model.py

**import psycopg2  
import time  
  
cursor = None  
connection = None  
  
  
def connect():  
 try:  
 global cursor, connection  
 connection = psycopg2.connect(  
 host="localhost",  
 user="postgres",  
 password="3214",  
 database="hostel",  
 port="5432"  
 )  
  
 cursor = connection.cursor()  
  
 print("Successfully CONNECTED to database HOSTEL")  
  
 except Exception as \_ex:  
 print("Failed CONNECTION to database HOSTEL", \_ex)  
  
  
def disconnect():  
 try:  
 cursor.close()  
 connection.close()  
 print("Successfully DISCONNECTED from database HOSTEL")  
 except Exception as \_ex:  
 print("Impossible to DISCONNECT from database HOSTEL", \_ex)  
  
  
def insert(choice: int, data: list) -> bool:  
 if connection is None or cursor is None:  
 return False  
 else:  
 try:  
 match choice:  
 case 1:  
 cursor.execute(f"""INSERT INTO public.\"dormitory\" (number, type, address, faculty) \  
 VALUES ({data[0]}, \'{data[1]}\', \'{data[2]}\', \'{data[3]}\');""")  
 case 2:  
 cursor.execute(f"""INSERT INTO public.\"room\_type\" (seats, furniture, gender) \  
 VALUES ({data[0]}, {data[1]}, \'{data[2]}\');""")  
 case 3:  
 cursor.execute(f"""INSERT INTO public.\"inmate\" (name, surname, gender, study\_year, dormitory\_id) \  
 VALUES (\'{data[0]}\', \'{data[1]}\', \'{data[2]}\', {data[3]}, {data[4]});""")  
 case 4:  
 cursor.execute(f"""INSERT INTO public.\"room\"(number, floor, settlement, eviction, dormitory\_id, \  
 room\_type\_id) VALUES({data[0]}, {data[1]}, \'{data[2]}\', \'{data[3]}\', {data[4]}, {data[5]});""")  
 case 5:  
 cursor.execute(f"""INSERT INTO public.\"room/inmate\" (room\_id, inmate\_id) \  
 VALUES ({data[0]}, {data[1]});""")  
 connection.commit()  
 except Exception as \_ex:  
 print(f"Impossible to INSERT data: {data}; into table with choice: {choice}", \_ex)  
 return False  
 return True  
  
  
def delete(table: str, key\_name: str, key\_val: str) -> bool:  
 if connection is None or cursor is None:  
 return False  
 else:  
 try:  
 cursor.execute(f"""DELETE FROM public.\"{table}\" WHERE {key\_name} = \'{key\_val}\';""")  
 connection.commit()  
 except Exception as \_ex:  
 print(f"Impossible to DELETE data from table {table} in database HOSTEL", \_ex)  
 return False  
 return True  
  
  
def select\_by\_key(table: str, key\_name: str, key\_val: str) -> list:  
 if connection is None or cursor is None:  
 return []  
 else:  
 try:  
 cursor.execute(f"""SELECT \* FROM public.\"{table}\" WHERE {key\_name} = \'{key\_val}\';""")  
 except Exception as \_ex:  
 print(f"Impossible to SELECT data from table {table} by key {key\_name} in database HOSTEL", \_ex)  
 return []  
 return cursor.fetchall()  
  
  
def select\_by\_table(table: str, quantity: str = '100', offset: str = '0') -> list:  
 if connection is None or cursor is None:  
 return []  
 else:  
 try:  
 if table == 'room/inmate':  
 cursor.execute(f"""SELECT \* FROM public.\"{table}\" ORDER BY {"room\_id"} \  
 ASC limit {quantity} offset {offset};""")  
 else:  
 cursor.execute(f"""SELECT \* FROM public.\"{table}\" ORDER BY {table + "\_id"} \  
 ASC limit {quantity} offset {offset};""")  
 except Exception as \_ex:  
 print(f"Impossible to SELECT data from table {table} in database HOSTEL", \_ex)  
 return []  
 return cursor.fetchall()  
  
  
def update(choice: int, data: list, id1: int, id2: int = 0) -> bool:  
 if connection is None or cursor is None:  
 return False  
 else:  
 try:  
 match choice:  
 case 1:  
 cursor.execute(f"""UPDATE public.\"dormitory\" SET number = {data[0]}, type = \'{data[1]}\', \  
 address = \'{data[2]}\', faculty = \'{data[3]}\' WHERE dormitory\_id = {id1};""")  
 case 2:  
 cursor.execute(f"""UPDATE public.\"room\_type\" SET seats = {data[0]}, furniture = {data[1]}, \  
 gender = \'{data[2]}\' WHERE room\_type\_id = {id1};""")  
 case 3:  
 cursor.execute(f"""UPDATE public.\"inmate\" SET name = \'{data[0]}\', surname = \'{data[1]}\', \  
 gender = \'{data[2]}\',study\_year = {data[3]}, dormitory\_id = {data[4]} WHERE inmate\_id = {id1};""")  
 case 4:  
 cursor.execute(f"""UPDATE public.\"room\" SET number = {data[0]}, floor = {data[1]}, \  
 settlement = \'{data[2]}\', eviction = \'{data[3]}\', dormitory\_id = {data[4]}, \  
 room\_type\_id = {data[5]} WHERE room\_id = {id1};""")  
 case 5:  
 cursor.execute(f"""UPDATE public.\"room/inmate\" SET room\_id = {data[0]}, \  
 inmate\_id = {data[1]} WHERE room\_id = {id1} AND inmate\_id = {id2};""")  
 connection.commit()  
 except Exception as \_ex:  
 print(f"Impossible to UPDATE data: {data}; into table with choice: {choice}", \_ex)  
 return False  
 return True  
  
  
def generate(choice: int, count: int) -> bool:  
 if connection is None or cursor is None:  
 return False  
 try:  
 for i in range(count):  
 match choice:  
 case 1:  
 cursor.execute(f"""INSERT INTO public.\"dormitory\" (number, type, address, faculty) \  
 VALUES ((floor(random() \* (30 - 1 + 1)) + 1), substr(md5(random()::text), 0, 8), \  
 substr(md5(random()::text), 0, 12), substr(md5(random()::text), 0, 5));""")  
 case 2:  
 cursor.execute(f"""INSERT INTO public.\"room\_type\" (seats, furniture, gender) \  
 VALUES ((floor(random() \* (4 - 1 + 1)) + 1), (round(random())::int)::boolean, \  
 substr(md5(random()::text), 0, 6));""")  
 case 3:  
 cursor.execute(f"""INSERT INTO public.\"inmate\" (name, surname, gender, study\_year, dormitory\_id) \  
 SELECT substr(md5(random()::text), 0, 10), \  
 substr(md5(random()::character varying(12)), 0, 12), \  
 substr(md5(random()::text), 0, 6), \  
 (floor(random() \* (2022 - 1975 + 1)) + 1975), \  
 dormitory\_id FROM public."dormitory" order by random() limit 1;""")  
 case 4:  
 cursor.execute(f"""INSERT INTO public.\"room\"(number, floor, settlement, eviction, dormitory\_id, \  
 room\_type\_id) \  
 SELECT floor(random() \* (1000 - 10 + 1) + 10), floor(random() \* (500 - 1 + 1) + 1),\  
 NOW() + (random() \* (NOW() - NOW() - '16000 days')), \  
 NOW() + (random() \* (NOW() - NOW() + '36000 days')), \  
 dormitory\_id, room\_type\_id FROM public."dormitory", public."room\_type" \  
 order by random() limit 1;""")  
 case 5:  
 cursor.execute(f"""INSERT INTO public.\"room/inmate\" (room\_id, inmate\_id) \  
 SELECT room\_id, inmate\_id FROM public."room", public."inmate" \  
 order by random() limit 1;""")  
 connection.commit()  
 except Exception as \_ex:  
 print("Impossible to GENERATE data to database HOSTEL", \_ex)  
 return False  
 return True  
  
  
def search(tables: list[str], key: str, value: str) -> tuple:  
 if connection is None or cursor is None:  
 return ()  
 try:  
 request = f"""SELECT \* FROM public.\"{tables[0]}\" as first INNER JOIN public.\"{tables[1]}\" as second on first.\"{key}\" = second.\"{key}\" WHERE {value}"""  
 print(f"SQL request: {request}")  
 start\_time = time.time\_ns()  
 cursor.execute(request)  
 rows = cursor.fetchall()  
 run\_time = time.time\_ns() - start\_time  
 except Exception as \_ex:  
 print("Impossible to SEARCH data in database HOSTEL", \_ex)  
 return ()  
 return rows, run\_time**

controller.py

**import model  
  
  
def input\_data(choice: int) -> list[str]:  
 print("Input data separated by comma")  
 data = ""  
 match choice:  
 case 1:  
 data = input('Table: dormitory. Input: number->int, type->text, address->text, faculty->text\n')  
 case 2:  
 data = input('Table: room\_type. Input: seats->int, furniture->boolean, gender->text\n')  
 case 3:  
 data = input('Table: inmate. Input: name->text, surname->text, gender->text, study\_year->int, dormitory\_id->int\n')  
 case 4:  
 data = input('Table: room. Input: number->int, floor->int, settlement->date, eviction->date, dormitory\_id->int, room\_type\_id->int\n')  
 case 5:  
 data = input('Table: room/inmate. Input: room\_id->int, inmate\_id->int\n')  
 return data.split(',')  
  
  
def print\_data(nums: list[int], rows):  
 d = []  
 for num in nums:  
 match num:  
 case 1:  
 d += ['dormitory\_id', 'number', 'type', 'address', 'faculty']  
 case 2:  
 d += ['room\_type\_id' ,'seats', 'furniture', 'gender']  
 case 3:  
 d += ['inmate\_id', 'name', 'surname', 'gender', 'study\_year', 'dormitory\_id']  
 case 4:  
 d += ['room\_id', 'number', 'floor', 'settlement', 'eviction', 'dormitory\_id', 'room\_type\_id']  
 case 5:  
 d += ['room\_id', 'inmate\_id']  
 names = []  
 lengths = []  
 rules = []  
 rls = []  
 for dd in d:  
 names.append(dd)  
 lengths.append(len(dd))  
 for col in range(len(lengths)):  
 for row in rows:  
 rls.append(3 if type(row[col]) is not str else len(row[col]))  
 lengths[col] = max([lengths[col]] + rls)  
 rules.append("#" \* lengths[col])  
 format = " ".join(["%%-%ss" % l for l in lengths])  
 result = [format % tuple(names), format % tuple(rules)]  
 for row in rows:  
 result.append(format % row)  
 return "\n".join(result) + '\n'  
  
  
def table\_by\_choice(choice: int) -> str:  
 table = ""  
 match choice:  
 case 1:  
 table = 'dormitory'  
 case 2:  
 table = 'room\_type'  
 case 3:  
 table = 'inmate'  
 case 4:  
 table = 'room'  
 case 5:  
 table = 'room/inmate'  
 return table  
  
  
def print\_request(choice: int, id: str = '', quantity: str = '0', offset: str = '0') -> str:  
 if choice <= 0 or choice > 5:  
 return ""  
 table = table\_by\_choice(choice)  
 if not id:  
 if quantity == '0':  
 quantity = str(input('Input quantity of rows to print: '))  
 rows = model.select\_by\_table(table, quantity, offset)  
 else:  
 if choice == 5:  
 rows = model.select\_by\_key(table, 'room\_id', id)  
 else:  
 rows = model.select\_by\_key(table, table + '\_id', id)  
 return print\_data([choice], rows)  
  
  
def select\_table(flag: bool = False) -> int:  
 choice = -1  
 if flag == True:  
 print("\n1. Generate data for all tables\n2. Generate data for one table")  
 choice = int(input('\nChoose the option: '))  
 while choice != 1 and choice != 2:  
 choice = int(input('\nError. Choose the option: '))  
 if choice == 1:  
 choice = 6  
 if flag == False or choice == 2:  
 print('1. dormitory\n2. room\_type\n3. inmate\n4. room\n5. room/inmate\n0. menu')  
 choice = int(input('\nChoose the table: '))  
 if choice > 6 or choice < 0:  
 print('Incorrect number, try one more time')  
 if flag == True:  
 select\_table(True)  
 else:  
 select\_table()  
 return choice  
  
  
def insert\_request(choice: int):  
 if choice <= 0 or choice > 6:  
 return  
 rows = [i.strip() for i in input\_data(choice)]  
 if model.insert(choice, rows):  
 print("Data INSERTED successfully")  
 else:  
 print("Impossible to insert data")  
  
  
def edit\_request(choice: int):  
 if choice <= 0 or choice > 5:  
 return  
 id = input("Enter id of row that you want to UPDATE\n"  
 "\'p\' => print rows\n\'r\' => return to menu\n")  
 id2 = ""  
 if choice == 5:  
 id2 = input("Enter id2 of row that you want to UPDATE\n")  
 if id == 'r':  
 return  
 elif id == 'p':  
 offset = '0'  
 while True:  
 print(print\_request(choice, quantity='15', offset=offset))  
 id = input("Enter id of row that you want to UPDATE\n"  
 "\'n\' => next 15 rows\n\'b\' => previous 15 rows\n\'r\' => return to menu\n")  
 if choice == 5:  
 id2 = input("Enter id2 of row that you want to UPDATE\n")  
 if id == 'r':  
 return  
 elif id == 'n':  
 offset = str(int(offset) + 15)  
 elif id == 'b':  
 offset = str(int(offset) - 15)  
 else:  
 break  
 print\_request(choice, id)  
 print("If you don't want to UPDATE column -> write as it was")  
 columns = input\_data(choice)  
 if choice == 5:  
 flag = model.update(choice, columns, int(id), int(id2))  
 else:  
 flag = model.update(choice, columns, int(id))  
 if flag:  
 print('UPDATED successfully')  
 else:  
 print("Impossible to UPDATE table")  
  
  
def delete\_request(choice: int):  
 if choice <= 0 or choice > 5:  
 return  
 table = table\_by\_choice(choice)  
 id = input("Enter id of row that you want to DELETE\n"  
 "\'p\' => print rows\n\'r\' => return to menu\n")  
 if id == 'r':  
 return  
 elif id == 'p':  
 offset = '0'  
 while True:  
 print(print\_request(choice, quantity='15', offset=offset))  
 id = input("Enter id of row that you want to DELETE\n"  
 "\'n\' => next 15 rows\n\'b\' => previous 15 rows\n\'r\' => return to menu\n")  
 if id == 'r':  
 return  
 elif id == 'n':  
 offset = str(int(offset) + 15)  
 elif id == 'b':  
 offset = str(int(offset) - 15)  
 else:  
 break  
 if choice == 5:  
 flag = model.delete(table, 'room\_id', id)  
 else:  
 flag = model.delete(table, table + '\_id', id)  
 if flag:  
 print('The row DELETED successfully')  
 else:  
 print("Impossible to DELETE the row")  
  
  
def generator\_request(choice: int):  
 if choice <= 0 or choice > 6:  
 return  
 quantity = int(input('Input the data quantity to GENERATE: '))  
 if choice == 6:  
 print("Data GENERATED and INSERTED into table dormitory successfully") \  
 if model.generate(1, quantity) else print("Impossible to GENERATE and INSERT data into table dormitory")  
 print("Data GENERATED and INSERTED into table room\_type successfully") \  
 if model.generate(2, quantity) else print("Impossible to GENERATE and INSERT data into table room\_type")  
 print("Data GENERATED and INSERTED into table inmate successfully") \  
 if model.generate(3, quantity) else print("Impossible to GENERATE and INSERT data into table inmate")  
 print("Data GENERATED and INSERTED into table room successfully") \  
 if model.generate(4, quantity) else print("Impossible to GENERATE and INSERT data into table room")  
 print("Data GENERATED and INSERTED into table room/inmate successfully") \  
 if model.generate(5, quantity) else print("Impossible to GENERATE and INSERT data into table room/inmate")  
 elif 0 < choice < 6:  
 print(f"Data GENERATED and INSERTED into table number {choice} successfully") \  
 if model.generate(choice, quantity) \  
 else print(f"Impossible to GENERATE and INSERT data into table number {choice}")  
  
  
def search\_request():  
 tables = []  
 tab = []  
 print('Choose the first table')  
 tab.append(select\_table())  
 tables.append(table\_by\_choice(tab[0]))  
 print('Choose the second table')  
 tab.append(select\_table())  
 tables.append(table\_by\_choice(tab[1]))  
 key = input('Input the connecting key: ')  
 print('Input the expression. Use "first" and "second" to address to the table attributes, if with string use like')  
 value = input()  
 rows = model.search(tables, key, value)  
 print("\n", print\_data(tab, rows[0]))  
 print('Time of the executing program:', rows[1] / 1000, ' milliseconds')  
  
  
def menu():  
 while True:  
 print('\n1. INSERT data in TABLE')  
 print('2. EDIT data in TABLE')  
 print('3. DELETE data from TABLE')  
 print('4. PRINT ROWS')  
 print('5. GENERATE random DATA')  
 print('6. SEARCH data from TABLES')  
 print('0. Exit')  
 match int(input('\t\t\tChoose an option 1-6 or 0: ')):  
 case 1:  
 insert\_request(select\_table())  
 case 2:  
 edit\_request(select\_table())  
 case 3:  
 delete\_request(select\_table())  
 case 4:  
 print("\n", print\_request(select\_table()))  
 case 5:  
 generator\_request(select\_table(True))  
 case 6:  
 search\_request()  
 case 0:  
 print("")  
 return**